Loops and Overloops for Tree-Walking Automata

Pierre-Cyrille Héam, Vincent Hugot, Olga Kouchnarenko

Loops and Overloops for Tree-Walking Automata

Pierre-Cyrille Héam[1], Vincent Hugot[2], Olga Kouchnarenko

FEMTO-ST CNRS 6174, University of Franche-Comté & INRIA/CASSIS, France

Abstract

Tree-Walking Automata (TWA) have lately received renewed interest thanks to their tight connection to XML. This paper introduces the notion of tree overloops, which is closely related to tree loops, and investigates the use of both for the following common operations on TWA: testing membership, transformation into a Bottom-Up Tree Automaton (BUTA), and testing emptiness. Notably, we argue that the transformation into a BUTA is slightly less straightforward than was previously assumed, show that using overloops yields much smaller BUTA in the deterministic case, and provide a polynomial over-approximation of this construction which detects emptiness with surprising accuracy against randomly generated TWA.

Keywords: Tree-Walking Automata, loops, overloops, membership, emptiness, approximation

1. Introduction

Tree-Walking Automata (TWA for short) are a well-established sequential model for recognising tree languages which was introduced in 1969 by Aho and Ullman [1]. While TWA originally received far less attention than the better known branching model of tree automata, they have been steadily gathering interest in the last few years. Notably, important questions which had remained open for decades have recently been closed. This renewed interest is owed in great part to the ever-growing popularity of XML, with which they and their variants are tightly connected, in particular through Core XPath [2] and streaming [3].

In this context, it becomes helpful to have reasonably efficient algorithms for essential operations on TWA such as deciding membership and emptiness, as

☆

Email addresses: pierre-cyrille.heam@inria.fr (Pierre-Cyrille Héam), vincent.hugot@inria.fr (Vincent Hugot), olga.kouchnarenko@inria.fr (Olga Kouchnarenko)

[1]This author is supported by the project ANR 2010 BLAN 0202 02 FREC.

[2]This author is supported by the French DGA (Direction Générale de l'Armement).

well as the transformation into a BUTA. Until now, research has been mainly focused on closing fundamental open problems concerning the expressiveness of TWA [4, 5, 6]. While algorithms for the above operations are known, they appear in print mostly as proof sketches, and there has been no focus on finding tighter complexity bounds. In contrast, this paper provides explicit algorithms for these tasks and deals with complexity issues. The common thread of our contributions is the notion of *tree loop*, which is pervasive to the algorithms we give. This notion is closely related to Knuth's construction for testing circularity of attribute grammars [7], and is a generalisation to trees of a similar construction for two-way word automata [8]. The contributions are organised as follows:

- ◇ Section 3.1[p4] gives a thorough introduction to tree loops – the basic idea of which is more or less folklore – and introduces a new notion of *tree overloop* in Sec. 3.3[p7]. Simple algorithms for testing membership follow naturally from this work; beyond the immediate application of the recursive definitions of loops and overloops, a more efficient method based on a Boolean matrix encoding of loops is given in Sec. 3.2[p6]. To the best of our knowledge, no such algorithm existed in the literature.

- ◇ Section 4 deals with the transformation from TWA to BUTA, based on the proof sketches in [9] and [10, p143]. Two variants are given in Sec. 4.1: one using loops and one using overloops. Section 4.2[p11] proceeds to show that, in the deterministic case, the overloops-based construction admits a much smaller upper bound on the number of generated states.

- ◇ The emptiness problem is known to be ExpTime-complete for TWA, and is traditionally tested by first transforming the TWA into a BUTA. Section 5 provides a polynomial-time algorithm which computes an "over-approximation" of this BUTA, and thus may decide emptiness positively. Should it prove inefficient against some families of TWA, then the approximation can be refined as much as needed.

- ◇ Section 6[p15] presents random experiments performed to confirm our theoretical results. They involve both an *ad-hoc* random generation scheme for non-deterministic TWA, and a more interesting one, based on the results of [11], which yields complete and deterministic TWA according to the uniform probability distribution – which imparts statistical significance to our results. The dependability of the approximation method developed in Sec. 5 is tested in Sec. 6.1 – it is shown to be astonishingly accurate against both schemes. Section 6.2[p16] compares the respective sizes of the BUTA obtained from the loops and overloops-based transformations, and shows that overloops yield much smaller BUTA than loops *in average*. It is also shown that this size gain is independent of (and cumulates with) post-processing cleanup (cf. [12]) of the BUTA. The ideas of these tests are illustrated on our running example, then validated against the above-mentioned uniform random generation scheme.

Note. This paper is an extended version of [13].

2. Preliminaries

Let $R \subseteq Q^2$ be a binary relation on a set Q; we denote by R^+ and R^* its transitive and reflexive-transitive closure, respectively. The notation $[\![n, m]\!]$ stands for the integer interval $\{ n, n+1, \ldots, m \}$.

We denote by $\mathbb{N}^*$ the set of words over $\mathbb{N}$; if $v, w \in \mathbb{N}^*$, then $v.w$ stands for the concatenation of the words v and w. A *binary alphabet* is a finite set of symbols, equipped with an arity function $\mathit{arity} : \Sigma \to \{0, 2\}$. The subset of symbols of Σ with arity k is denoted by Σ_k. The set $\mathcal{T}(\Sigma)$ of (binary) trees over Σ is defined inductively as the smallest set such that $\Sigma_0 \subseteq \mathcal{T}(\Sigma)$ and, if $f \in \Sigma_2$ and $u_0, u_1 \in \mathcal{T}(\Sigma)$, then $f(u_0, u_1) \in \mathcal{T}(\Sigma)$. If $t \in \mathcal{T}(\Sigma)$ is a tree, then the set of *positions* (or *nodes*) $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ is defined inductively by $\mathcal{P}os(t) = \{\varepsilon\}$ if t is a constant (i.e. $t \in \Sigma_0$) and $\mathcal{P}os\big(f(u_0, u_1)\big) = \{\varepsilon\} \cup \{ k.\alpha_k \mid k \in [\![0, 1]\!] \text{ and } \alpha_k \in \mathcal{P}os(u_k) \}$ otherwise. We see a tree t as a function $t : \mathcal{P}os(t) \to \Sigma$ which maps a position to the symbol at that position in t. Positions are equipped with a non-strict (resp. strict) partial order $\trianglelefteq$ (resp. $\triangleleft$), such that $\alpha \trianglelefteq \beta$ iff β is a prefix of α (resp. $\alpha \trianglelefteq \beta$ and $\alpha \neq \beta$). The *size* of a tree t is denoted by $\|t\|$ and defined by $\|t\| = |\mathcal{P}os(t)|$. The *parent function* $\mathfrak{p}(\cdot) : \mathcal{P}os(t) \setminus \{\varepsilon\} \to \mathcal{P}os(t)$ maps any (non-root) *child* node $\alpha.k$ (where $k \in \{0, 1\}$) to its *father* α. We denote by $t|_\alpha$ the subtree of t under α. The reader is assumed to be well-acquainted with the bottom-up variety of branching tree automata (see for instance [14]).

A Tree-Walking Automaton (TWA) can be thought of intuitively as a head moving in the tree from father to son and from son to father. The head chooses its next move based on its internal state, the symbol at its current position, and whether its current position is the root of the tree, a left son, or a right son. A TWA accepts a tree if, starting from the root in an initial state, its head can move back to the root in a final state.

Formally, a TWA is a tuple $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ where Q is a finite set of states, Σ a binary alphabet, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ a subset of final – or accepting – states, and

$$\Delta \subseteq \Sigma \times Q \times \underbrace{\{ \star, \boldsymbol{0}, \boldsymbol{1} \}}_{\mathbb{T}\,:\,\text{types}} \times \underbrace{\{ \uparrow, \circlearrowleft, \swarrow, \searrow \}}_{\mathbb{M}\,:\,\text{moves}} \times Q$$

is a set of transitions. In this paper the tuple $\langle \Sigma, Q, I, F, \Delta \rangle$ will be assumed whenever we speak of a TWA $\mathcal{A}$. Each node α of a tree t has a *type* in $\mathbb{T}$, denoted by $\natural\alpha$, such that $\natural\varepsilon = \star$ (root), $\natural(\beta.0) = \boldsymbol{0}$ (left son), $\natural(\beta.1) = \boldsymbol{1}$ (right son). As we will seldom deal with the root in practice, we define for short the *sons* $\mathbb{S} = \{ \boldsymbol{0}, \boldsymbol{1} \} \subset \mathbb{T}$. We will also put in relation types and moves through the function $\chi(\cdot) : \mathbb{S} \to \{ \swarrow, \searrow \}$ such that $\chi(\boldsymbol{0}) = \swarrow$ and $\chi(\boldsymbol{1}) = \searrow$. For our convenience, we will take the special notation $\langle f, p, \tau \to \mu, q \rangle$ for the tuple $(f, p, \tau, \mu, q) \in \Delta$. Some of the parameters can be replaced by sets, with the obvious meaning that we consider the set of all transitions thus described. For instance $\langle \Sigma_2, p, \mathbb{T} \to \circlearrowleft, q \rangle = \{ (\sigma, p, \tau, \circlearrowleft, q) \mid \sigma \in \Sigma_2, \tau \in \mathbb{T} \}$. Note that all the transitions from $\langle \Sigma_0, Q, \mathbb{T} \to \{ \swarrow, \searrow \}, Q \rangle \cup \langle \Sigma, Q, \star \to \uparrow, Q \rangle$ are invalid.

A *configuration* of $\mathcal{A}$ on a tree t is a pair $c = (\beta, q) \in \mathcal{P}os(t) \times Q$; it is *initial* if $c \in \{\varepsilon\} \times I$ and *final* (or *accepting*) if $c \in \{\varepsilon\} \times F$. It is a *successor* of a

configuration (α, p) if $\langle t(\alpha), p, \natural\alpha \to \mu, q\rangle \in \Delta$, where μ is $\uparrow$ if $\beta = \mathfrak{p}(\alpha)$, $\circlearrowleft$ if $\beta = \alpha$, $\swarrow$ if $\beta = \alpha.0$ and $\searrow$ if $\beta = \alpha.1$. We write $c_1 \twoheadrightarrow_{\mathcal{A}} c_2$ (or simply $c_1 \twoheadrightarrow c_2$ whenever $\mathcal{A}$ is clear from the context) if the configuration c_2 is a successor of c_1. A *run* is a sequence of successive configurations $c_1 \twoheadrightarrow c_2 \twoheadrightarrow \ldots c_n \twoheadrightarrow \ldots$. A run is *accepting* (or *successful*) if it starts with an initial configuration and reaches a final configuration. A tree t is *accepted* or *recognised* by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on t. The set of all accepted trees is the *language* of $\mathcal{A}$, denoted by $\mathcal{L}\mathrm{ng}(\mathcal{A})$.

Example: Let $\mathcal{X}$ be a TWA such that $\Sigma_0 = \{a, b, c\}$ and $\Sigma_2 = \{f, g, h\}$, $Q = \{q_\ell, q_\mathrm{u}\}$, $I = \{q_\ell\}$, $F = \{q_\mathrm{u}\}$, and $\Delta = \langle a, q_\ell, \{\star, \boldsymbol{0}\} \to \circlearrowleft, q_\mathrm{u}\rangle \cup \langle \Sigma, q_\mathrm{u}, \boldsymbol{0} \to \uparrow, q_\mathrm{u}\rangle \cup \langle \Sigma_2, q_\ell, \{\star, \boldsymbol{0}\} \to \swarrow, q_\ell\rangle$. Then $\mathcal{X}$ accepts exactly all trees whose left-most leaf is labelled by a. We shall use this (trivial) example throughout the paper. $\mathcal{X}$ has two states and fourteen rules.

3. Loops, Overloops and the Membership Problem

3.1. Defining, Classifying and Computing Loops

The notion of *loop* turned out to be very useful to deal with TWA. Informally, loops arise naturally as a generalisation of the definition of an accepting run, where the automaton enters the root in a given initial state p_in, moves along the tree, and then comes back to the root in a certain final state p_out. In practice, the details of the moves which form the loop itself are largely irrelevant and are discarded: the most useful information is the pair of states $(p_\mathrm{in}, p_\mathrm{out})$.

Definition 1 (Tree Loops)**.** Let $\mathcal{A}$ be a TWA, t a tree and $\alpha \in \mathcal{P}os(t)$. A pair of states $(p, q) \in Q^2$ is a *loop* of $\mathcal{A}$ on the subtree $t|_\alpha$ if there exist $n \geqslant 0$ and a run $(\alpha, p), (\beta_1, s_1), \ldots, (\beta_n, s_n), (\alpha, q)$ such that $\beta_k \trianglelefteq \alpha$ for all $k \in [\![1, n]\!]$. Such a run is a *looping run*, and we say that it *forms* the loop (p, q).

Example: The looping run $(0, q_\ell), (0.0, q_\ell), (0.0, q_\mathrm{u}), (0, q_\mathrm{u})$ of $\mathcal{X}$ on the subtree $g(f(a, b), c)|_0 = f(a, b)$ forms the loop (q_ℓ, q_u).

Notice that loops are not only defined on whole trees, but on subtrees as well with the restriction that the automaton cannot leave the subtree during the looping run. It is in fact this restriction which grants loops their usefulness. TWA, unlike their branching cousins, whose runs are defined inductively, do not naturally lend themselves to inductive reasoning; and yet, thanks to the above restriction, loops are easily computed by induction. Thus loops and their variants can be thought of as convenient devices which hide the sequential, stateful aspect of TWA runs beneath a much more "user-friendly" layer of induction.

In the next few paragraphs we compute the loops of a TWA $\mathcal{A}$ on a subtree $t|_\alpha$.

Definition 2 (Kinds of Loops)**.** Clearly for all $p \in Q$, (p, p) is a loop; we call such loops *trivial*. A looping run of $\mathcal{A}$ on $t|_\alpha$ is *simple* if it reaches α exactly twice. It is *non-trivial* if it reaches α at least twice. A loop is *simple* (resp. *non-trivial*) if there exists a *simple* (resp. *non-trivial*) looping run forming it.

Example: The loop (q_ℓ, q_u) in the above example is simple, because $(0, q_\ell), (0.0, q_\ell), (0.0, q_\mathrm{u}), (0, q_\mathrm{u})$ only reaches $\alpha = 0$ twice, on the first and last configuration. The TWA $\mathcal{X}$ forms only trivial and simple loops, but suppose that we alter it so that it also checks that the *right-most* leaf is a. During an accepting run it would go down and left, back up to the root, down and right, and back up to the root again, in a final state. Thus all accepting runs would be non-trivial and non-simple, reaching the root exactly three times.

Fortunately, we only ever need to compute simple loops, as we can deduce the rest from them thanks to the following lemma:

Lemma 3 (*Loop Decomposition*)**.** *If $S \subseteq Q^2$ is the set of all simple loops of $\mathcal{A}$ on a given subtree $u = t|_\alpha$, then S^* is the set of all loops of $\mathcal{A}$ on u.*

Proof. Every looping run is either trivial or non-trivial. All trivial loops are in S^* by the reflexive closure. Furthermore, every non-trivial looping run can easily be decomposed into one or more simple runs. Indeed, any non-trivial looping run ℓ has the following general form, where $\beta_i^k \lhd \alpha$ for all k, i, and the notation $[x_k]^{k\in[\![1,m]\!]}$ designates the run obtained by concatenating the runs $x_1, \dots, x_m$:

$$\ell = (\alpha, p^0), \left[(\beta_1^k, s_1^k), \dots, (\beta_{n_k}^k, s_{n_k}^k), (\alpha, p^k)\right]^{k\in[\![1,m]\!]} \quad .$$

This is the composition of m simple looping runs ℓ_k, for $k \in [\![1, m]\!]$, forming the simple loops (p^{k-1}, p^k). The remaining loops are obtained by transitive closure:

$$\left\{ (p^{k-1}, p^k) \;\middle|\; k \in [\![1, m]\!] \right\}^+ = \left\{ (p^{k-1}, p^l) \;\middle|\; k, l \in [\![1, m]\!],\ k \leqslant l \right\} . \qquad \square$$

Let us denote by $\mho^\tau(u)$ the set of all loops of $\mathcal{A}$ on a subtree u, where τ is the type of the root of u; if u is the subtree $t|_\alpha$ then $\tau = \natural\alpha$. Note that thanks to the above-mentioned restriction in the definition of loops, the type of the subtree's root is the only information which is actually needed from the context.

Let $a \in \Sigma_0$ be a leaf of type τ. We compute the loops on a. By definition of a looping run, $\mathcal{A}$ cannot move up; nor can it move down since leaves have no children. So the only transitions which can be activated are $\circlearrowleft$-transitions. As we are only interested in *simple* loops, we can only activate one of these transitions *once*, thus creating runs of the form $(\alpha, p) \twoheadrightarrow (\alpha, q)$, and the corresponding loops (p, q). Let us have a general notation for this:

Definition 4 (Simple Here-Loops)**.** $\mathcal{H}_\sigma^\tau \triangleq \{ (p, q) \mid \langle \sigma, p, \tau \to \circlearrowleft, q\rangle \in \Delta \}$.

Thus the simple loops on a are $\mathcal{H}_a^\tau$. By Lemma 3 we have $\mho^\tau(a) = (\mathcal{H}_a^\tau)^*$. We now deal with inner nodes. Let $f \in \Sigma_2$, and $u = f(u_{\mathit{0}}, u_{\mathit{1}})$; again, τ denotes the type of the root of u. Clearly the elements of $\mathcal{H}_f^\tau$ are loops on u, as above, but this time $\mathcal{A}$ can move down as well. It cannot move up on the first move (that would mean leaving the subtree), but it will obviously *need* to move up to rejoin the root if it ever moves down. To clarify all that, let us reason on what the first move of a simple looping run can be. It cannot be $\uparrow$ and all simple loops whose first move is $\circlearrowleft$ are already computed in $\mathcal{H}_f^\tau$. Say the first move is $\swarrow$: then the run can do

whatever it wants in the left subtree $u_{\boldsymbol{0}}$, after which it has to move back up to the root to complete the loop. Again, we only consider *simple* loops, so no move can be made past this point, as the root has been reached twice already. Thus the general form of such a run is $(\varepsilon, p), (0, p_{\boldsymbol{0}}), (\beta_1, s_1), \ldots, (\beta_n, s_n), (0, q_{\boldsymbol{0}}), (\varepsilon, q)$, with all $\beta_k \trianglelefteq 0$. But by definition, this means that $(p_{\boldsymbol{0}}, q_{\boldsymbol{0}})$ is a loop on $u_{\boldsymbol{0}}$, i.e. $(p_{\boldsymbol{0}}, q_{\boldsymbol{0}}) \in \mho^{\boldsymbol{0}}(u_{\boldsymbol{0}})$. Needless to say, the same applies (with $\boldsymbol{1}$ instead of $\boldsymbol{0}$) if the first move is $\searrow$. It follows that to determine whether (p, q) forms a simple loop on u, we need only check three things: (1) $\mathcal{A}$ can move down (left or right) from state p into a state p_θ, (2) there is a loop (p_θ, q_θ) on this subtree and (3) in state q_θ, $\mathcal{A}$ can move up from this subtree and into the state q. Formally:

$$\mho^\tau(u) = \left(\mathcal{H}^\tau_f \cup \left\{ (p,q) \;\middle|\; \begin{array}{c} \exists \theta \in \mathbb{S} : \\ \exists (p_\theta, q_\theta) \in \mho^\theta(u_\theta) \end{array} \text{ st. } \begin{array}{l} \langle f, p, \tau \to \chi(\theta), p_\theta \rangle \in \Delta \\ \langle u_\theta(\varepsilon), q_\theta, \theta \to \uparrow, q \rangle \in \Delta \end{array} \right\} \right)^* .$$

Theorem 5 (*Loops*)**.** *Let $\mathcal{A}$ be a TWA and $t \in \mathcal{T}(\Sigma)$. Then for all $\alpha \in \mathcal{P}os(t)$, $\mho^{\natural\alpha}(t|_\alpha)$, as defined above, is the set of all loops of $\mathcal{A}$ on $t|_\alpha$.*

Example: For the TWA $\mathcal{X}$, $\mho^{\boldsymbol{0}}(a) = \{ (q_\ell, q_\mathrm{u}) \}^* = \{ (q_\ell, q_\ell), (q_\mathrm{u}, q_\mathrm{u}), (q_\ell, q_\mathrm{u}) \}$, and $\mho^\star(f(a,b)) = (\varnothing \cup \{(q_\ell, q_\mathrm{u})\})^*$ (no simple here-loop, and one loop built on the left child). On the other hand, $\mho^\star(f(b,a)) = \varnothing^*$, because $\mho^{\boldsymbol{1}}(a) = \mho^{\boldsymbol{0}}(b) = \varnothing^*$.

3.2. A Direct Application of Loops to Membership Testing

Note that a reasonably efficient algorithm for testing membership is straightforwardly derived from the above computation of loops:

Corollary 6 (*TWA Membership*)**.** *Let $\mathcal{A}$ be a TWA and $t \in \mathcal{T}(\Sigma)$. Then we have $t \in \mathcal{L}ng(\mathcal{A})$ if and only if $\mho^\star(t) \cap (I \times F) \neq \varnothing$.*

Proof. There is a loop $(q_\mathrm{i}, q_\mathrm{f}) \in I \times F$ of $\mathcal{A}$ on t iff there is a run of the form $(\varepsilon, q_\mathrm{i}), \ldots, (\varepsilon, q_\mathrm{f})$. The first configuration is initial, and the last final. Therefore it is an accepting run, and $t \in \mathcal{L}\mathrm{ng}(\mathcal{A})$. □

Corollary 7. *The complexity of TWA membership is $O\left(|\Delta| + \|t\| \cdot |Q|^3\right)$.*

Proof. A naïve computation of $\mho^\star(t)$ would be done in $O\left(\|t\| \cdot (|Q|^3 + |Q|^2 \cdot |\Delta|)\right)$. The following algorithm, while still simple, runs in $O\left(|\Delta| + \|t\| \cdot |Q|^3\right)$, at the cost of a $O(\|t\| \cdot |Q|^2)$ space complexity. **Preliminaries.** Transitions and loops will be represented by relations from Q to Q, coded as matrices of $\mathcal{M}_{|Q|}(\mathbb{B})$ within the classical Boolean algebra $(\mathbb{B}, +, \cdot)$. The states of Q are numbered and assimilated to their indices $[\![1, n]\!]$ for the sake of denotational simplicity. A relation $R \subseteq Q^2$ is represented by the matrix $\mathcal{M}[R] = (\mathcal{M}[R]_{ij})$, such that $\mathcal{M}[R]_{ij} = 1$ iff jRi. The sum and product of matrices are defined as usual. With those conventions we have the expected result regarding composition: let $R, R' \subseteq Q^2$ and $P = \mathcal{M}[R'] \times \mathcal{M}[R]$; then $P_{ij} = \sum_{k=1}^n \mathcal{M}[R']_{ik} \mathcal{M}[R]_{kj}$. Thus $P_{ij} = 1$ iff there exists k such that jRk and $kR'i$, that is to say, $j(R' \circ R)i$. In other words $\mathcal{M}[R' \circ R] = \mathcal{M}[R'] \times \mathcal{M}[R]$. **Input & Variables.** A TWA $\mathcal{A}$ and a tree t form the input. The core of the algorithm is the sub-function f, which

takes as input α (a position in $\mathcal{P}os(t)$). Its call defines a matrix L^α, representing the loops at position α. **Algorithm.** Initialisation. For each $\sigma \in \Sigma$, $\tau \in \mathbb{T}$, $\mu \in \mathbb{M}$, a matrix $T^{\sigma,\tau,\mu}$ is built such that $T^{\sigma,\tau,\mu}_{qp} = 1$ iff $\langle \sigma, p, \tau \to \mu, q\rangle \in \Delta$. The positions of $\mathcal{P}os(t)$ are topologically ordered as $\alpha_1, \ldots, \alpha_m = \varepsilon$. Body. For $k = 1$ to m, $f(\alpha_k)$ is called. Then L^ε is returned. On a call to $f(\alpha)$: **(1)** Populate the matrix

$$L^\alpha = T^{t(\alpha),\natural\alpha,\circlearrowleft} + \sum_{\theta\in\mathbb{S}} \left[T^{t(\alpha.\theta),\natural(\alpha.\theta),\uparrow} \times L^{\alpha.\theta} \times T^{t(\alpha),\natural\alpha,\chi(\theta)} \right].$$

(2) Compute the reflexive and transitive closure of L^α in place. **Complexity.** The initial topological sorting is done in $O(\|t\|)$, and the construction of the $T^{\sigma,\tau,\mu}$ matrices is done in $O(|\Sigma| \cdot |Q|^2 + |\Delta|)$. Within each call of f we have the following complexities: (1) $O(|Q|^{2.3727})$ using the latest Coppersmith–Winograd–Stothers–Williams algorithm — or simply $O(|Q|^3)$ with the conventional product (2) $\Theta(|Q|^3)$ using the Roy–Floyd–Warshall algorithm. The complexity of any call to f is therefore $O(|Q|^3)$; there are $\|t\|$ calls to f. Hence the announced total complexity of $O\left(|\Delta| + \|t\| \cdot |Q|^3\right)$. **Correctness.** After the call to $f(\alpha)$, it is plain that L^α encodes $\mho^{\natural\alpha}(t|_\alpha)$, as the computation of (1) and (2) is a straight-forward reformulation of the formula of Thm. 5[p6] in terms of a Boolean matrix representation. The recursive nature of that formula has been unwound in this algorithm by the prior topological sorting of the positions. □

3.3. From Loops to Overloops

We now introduce a new notion related to tree loops: *tree overloops*. An overloop is formed by a looping run followed by a move up; this apparently minor change has a number of positive consequences which we discuss in the next sections.

Definition 8 (Over-Root, Extended Positions and Transitions)**.** The *extended positions* $\overline{\mathcal{P}os}(t)$ of a tree $t \in \mathcal{T}(\Sigma)$ are the set $\mathcal{P}os(t) \cup \{\overline{\varepsilon}\}$, where $\overline{\varepsilon}$ is called the *overroot*. The parent function $\mathfrak{p}(\cdot)$ is extended over $\overline{\mathcal{P}os}(t)$ into the *extended parent function* $\overline{\mathfrak{p}}(\cdot)$, such that $\overline{\mathfrak{p}}(\varepsilon) = \overline{\varepsilon}$ and $\varepsilon \lhd \overline{\varepsilon}$. The notion of configuration is extended as well, so that the transitions of $\langle \Sigma, Q, \star \to \uparrow, Q\rangle$ become valid. Their application yields configurations of the form $(\overline{\varepsilon}, q)$.

Definition 9 (Tree Over-Loops)**.** Let $\mathcal{A}$ be a TWA and t a tree. A pair of states $(p, q) \in Q^2$ forms an *overloop* of $\mathcal{A}$ on $t|_\alpha$ if there exists a run $(\alpha, p), (\beta_1, s_1), \ldots, (\beta_n, s_n), (\overline{\mathfrak{p}}(\alpha), q)$ such that $\beta_k \trianglelefteq \alpha$ for all $k \in [\![1, n]\!]$.

A way to compute overloops is to compute loops, then check for $\uparrow$-transitions:

Definition 10 (Up-Closure)**.** Let $L \subseteq Q^2, \tau \in \mathbb{T}$ and $\sigma \in \Sigma$:

$$\mathcal{U}^\tau_\sigma[L] \triangleq \left\{ (p, q) \in Q^2 \;\middle|\; \exists p' \in Q : (p, p') \in L \text{ and } \langle \sigma, p', \tau \to \uparrow, q\rangle \in \Delta \right\} .$$

Lemma 11 (*Up-Closure*)*. Let $\mathcal{A}$ be a TWA. If L is the set of all loops of $\mathcal{A}$ on a subtree $u = t|_\alpha$, then $\mathcal{U}^{\natural\alpha}_{t(\alpha)}[L]$ is the set of all overloops of $\mathcal{A}$ on u.*

Proof. Immediate from Def. 9, as we have necessarily $\beta_n = \alpha$. Thus any overloop is a loop followed by a move up, and conversely. □

Similarly to loops, we denote by $\mathfrak{O}^\tau(u)$ the set of all overloops of $\mathcal{A}$ on a subtree u, where τ is the type of the root of u. By Lem. 11 we have $\mathfrak{O}^\tau(u) = \mathcal{U}^\tau_{u(\varepsilon)}[\mho^\tau(u)]$, and in the case of leaves this yields $\mathfrak{O}^\tau(a) = \mathcal{U}^\tau_a[(\mathcal{H}^\tau_a)^*]$. However, in the case of inner nodes (say $u = f(u_0, u_1)$), in order to have an inductive computation of overloops instead of one based on loops, we need to compute the overloops of the father, knowing the overloops of the children. The simplest way is to compute the loops of the father and take the up-closure. We only need to check whether (1) the automaton can go down and left (resp. right) from p to a state p_0 and (2) there is a left (resp. right) overloop (p_0, q_0): this forms a loop (p, q_0). Formally:

$$\mathfrak{O}^\tau(u) = \mathcal{U}^\tau_f\left[\left(\mathcal{H}^\tau_f \cup \left\{ (p, q_\theta) \;\middle|\; \begin{array}{l} \exists \theta \in \mathbb{S} : \\ \exists p_\theta \in Q \end{array} \text{ st. } \begin{array}{l} \langle f, p, \tau \to \chi(\theta), p_\theta \rangle \in \Delta \\ \text{and } (p_\theta, q_\theta) \in \mathfrak{O}^\theta(u_\theta) \end{array} \right\}\right)^*\right] .$$

Theorem 12 (*Overloops*)**.** *Let $\mathcal{A}$ be a TWA and $t \in \mathcal{T}(\Sigma)$. Then for all $\alpha \in \mathcal{P}os(t)$, $\mathfrak{O}^{\natural\alpha}(t|_\alpha)$, as defined above, is the set of all overloops of $\mathcal{A}$ on $t|_\alpha$.*

Example: For the TWA $\mathcal{X}$, $\mathfrak{O}^0(a) = \mathcal{U}^0_a\big[\mho^0(a)\big] = \{(q_u, q_u), (q_\ell, q_u)\}$. However $\mho^\star(f(a,b))$ is the empty set. Thus a small adjustment is needed to test membership using overloops, as standard TWA – such as $\mathcal{X}$ – never admit any overloop at the root of a tree, for the lack of ↑-transitions.

Definition 13 (Overfinal State & Escaped TWA)**.** Let $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ be a TWA; it can be transformed into an *escaped TWA*

$$\mathcal{A}' = \big\langle\ \Sigma,\ Q \uplus \{\checkmark\},\ I, F,\ \Delta \uplus \langle \Sigma, F, \star \to \uparrow, \checkmark \rangle\ \big\rangle\ ,$$

where $\checkmark \notin Q$ is a fresh state, called *overfinal state*. [Clearly $\mathcal{L}\mathrm{ng}(\mathcal{A}) = \mathcal{L}\mathrm{ng}(\mathcal{A}')$.]

Example: Once $\mathcal{X}$ is escaped, we have $\mathfrak{O}^\star(f(a,b)) = \{ (q_u, \checkmark), (q_\ell, \checkmark) \}$.

Corollary 14 (*TWA Membership Redux*)**.** *Let $\mathcal{A}$ be an escaped TWA and $t \in \mathcal{T}(\Sigma)$. Then $t \in \mathcal{L}ng(\mathcal{A})$ if and only if $\mathfrak{O}^\star(t) \cap (I \times \{\checkmark\}) \neq \varnothing$.*

Proof. The couple $(q_i, \checkmark) \in I \times \{\checkmark\}$ is an overloop iff there is a run $(\varepsilon, q_i), \ldots, (\varepsilon, q_f), (\bar{\varepsilon}, \checkmark)$. By Def. 13, we must have $q_f \in F$; therefore, by Cor. 6 we have immediately $t \in \mathcal{L}\mathrm{ng}(\mathcal{A})$. □

4. Transforming TWA into equivalent BUTA

It is well-known that every TWA is equivalent to a BUTA; a more general version of this result has been proven in [15] – using game-theoretic arguments – and the main idea of a loop-based transformation from TWA into BUTA is outlined in [9] and [10, p143]. In this section we present two versions of it: the classical, loop-based one (Algo. $2_{[p9]}$) and an overloop-based variant (Algo. $3_{[p9]}$). Since those algorithms share a strong common structure, they are given as

Data: A TWA $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$
Input: $\langle P_{\text{init}} \rangle$, $\langle P_{\boldsymbol{0}} \rangle$, $\langle P_{\boldsymbol{1}} \rangle$, $\langle P_{\text{indu}} \rangle$, $\langle F \rangle$
Result: A BUTA $\mathcal{B}$

initialise States and Rules to $\varnothing$
foreach $a \in \Sigma_0, \tau \in \mathbb{T}$ **do**
A add $a \to \langle P_{\text{init}} \rangle$ to Rules and $\langle P_{\text{init}} \rangle$ to States
repeat
  foreach $f \in \Sigma_2, \tau \in \mathbb{T}$ **do**
B add every $f(\langle P_{\boldsymbol{0}} \rangle, \langle P_{\boldsymbol{1}} \rangle) \to \langle P_{\text{indu}} \rangle$ to Rules and $\langle P_{\text{indu}} \rangle$ to States
    where $\langle P_{\boldsymbol{0}} \rangle, \langle P_{\boldsymbol{1}} \rangle \in$ States
until Rules *remains unchanged*
return $\mathcal{B} = \langle \Sigma, \text{States}, \langle F \rangle, \text{Rules} \rangle$

Algorithm 1: Meta-Transformation into BUTA

Data: A TWA $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$
Result: A BUTA $\mathcal{B}$ such that $\mathcal{L}\text{ng}(\mathcal{B}) = \mathcal{L}\text{ng}(\mathcal{A})$

Meta-Algorithm 1 **where**

$$\langle P_{\text{init}} \rangle \equiv (a, \tau, \mathcal{H}_a^{\tau *}) \qquad \langle P_{\text{indu}} \rangle \equiv (f, \tau, (\mathcal{H}_f^{\tau} \cup S)^*)$$
$$\langle P_{\boldsymbol{0}} \rangle \equiv (\sigma_{\boldsymbol{0}}, \boldsymbol{0}, S_{\boldsymbol{0}}) \qquad \langle P_{\boldsymbol{1}} \rangle \equiv (\sigma_{\boldsymbol{1}}, \boldsymbol{1}, S_{\boldsymbol{1}})$$
$$\langle F \rangle \equiv \{\, (\sigma, \star, L) \in \text{States} \mid L \cap (I \times F) \neq \varnothing \,\}$$

$$S = \left\{ (p, q) \;\middle|\; \exists \theta \in \mathbb{S}, (p_\theta, q_\theta) \in S_\theta : \;\middle|\; \begin{array}{l} \langle f, p, \tau \to \chi(\theta), p_\theta \rangle \in \Delta \text{ and} \\ \langle \sigma_\theta, q_\theta, \theta \to \uparrow, q \rangle \in \Delta \end{array} \right\}$$

Algorithm 2: Tranformation into BUTA, with loops

Data: An escaped TWA $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ (see Def. 13)
Result: A BUTA $\mathcal{B}$ such that $\mathcal{L}\text{ng}(\mathcal{B}) = \mathcal{L}\text{ng}(\mathcal{A})$

Meta-Algorithm 1 **where**

$$\langle P_{\text{init}} \rangle \equiv (\tau, \mathcal{U}_a^{\tau}[\mathcal{H}_a^{\tau *}]) \qquad \langle P_{\text{indu}} \rangle \equiv (\tau, \mathcal{U}_f^{\tau}[(\mathcal{H}_f^{\tau} \cup S)^*])$$
$$\langle P_{\boldsymbol{0}} \rangle \equiv (\sigma_{\boldsymbol{0}}, S_{\boldsymbol{0}}) \qquad \langle P_{\boldsymbol{1}} \rangle \equiv (\sigma_{\boldsymbol{1}}, S_{\boldsymbol{1}})$$
$$\langle F \rangle \equiv \{\, (\star, O) \in \text{States} \mid O \cap (I \times \{\checkmark\}) \neq \varnothing \,\}$$

$$S = \left\{ (p, q_\theta) \;\middle|\; \exists \theta \in \mathbb{S}, p_\theta \in Q : \;\middle|\; \begin{array}{l} \langle f, p, \tau \to \chi(\theta), p_\theta \rangle \in \Delta \\ \text{and } (p_\theta, q_\theta) \in S_\theta \end{array} \right\}$$

Algorithm 3: Tranformation into BUTA, with overloops

instantiations of Meta-Algorithm 1, whose inputs (between angle brackets $\langle\cdot\rangle$) are substituted into its body. We go on to show that, in the case of deterministic TWA, the overloop-based construction results in much smaller equivalent BUTA than the classical one.

4.1. Two Variants: Loops and Overloops

Lemma 15 (*Loop-Based Algorithm*)**.** *Let $\mathcal{A}$ be a TWA, $\mathcal{B}$ the BUTA constructed by Algorithm 2, $t \in \mathcal{T}(\Sigma)$ and a position $\alpha \in \mathcal{P}os(t)$. Then for every type $\tau \in \mathbb{T}$ there is a unique run ρ of $\mathcal{B}$ on $t|_\alpha$, which is such that $\rho(\varepsilon) = (t(\alpha), \tau, \mho^\tau(t|_\alpha))$.*

Proof. By structural induction on $u = t|_\alpha$. Base Case: $u = a \in \Sigma_0$. By line A in Algorithm 2, $\rho(\varepsilon) = P = (a, \tau, \mathcal{H}_a^{\tau *}) = (t(\alpha), \tau, \mathcal{H}_a^{\tau *})$. This is the only possible run, as only one transition $a \to P$ is generated for each couple a, τ. By Theorem 5 we have $\mathcal{H}_a^{\tau *} = \mho^\tau(a)$. Inductive Case: $u = f(u_{\boldsymbol{0}}, u_{\boldsymbol{1}})$. By induction hypothesis the run $\rho_{\boldsymbol{0}}$ on $u_{\boldsymbol{0}}$ is such that $\rho_{\boldsymbol{0}}(\varepsilon) = P_{\boldsymbol{0}} = (u_{\boldsymbol{0}}(\varepsilon), \boldsymbol{0}, \mho^{\boldsymbol{0}}(u_{\boldsymbol{0}}))$, and the run $\rho_{\boldsymbol{1}}$ on $u_{\boldsymbol{1}}$ is such that $\rho_{\boldsymbol{1}}(\varepsilon) = P_{\boldsymbol{1}} = (u_{\boldsymbol{1}}(\varepsilon), \boldsymbol{1}, \mho^{\boldsymbol{1}}(u_{\boldsymbol{1}}))$. By line B in Algo. 2 we use the rule $f(P_{\boldsymbol{0}}, P_{\boldsymbol{1}}) \to P$ to build a run ρ such that $\rho(\varepsilon) = P = (f, \tau, (\mathcal{H}_f^\tau \cup S)^*) = (u(\varepsilon), \tau, (\mathcal{H}_f^\tau \cup S)^*)$, $\rho|_{\boldsymbol{0}} = \rho_{\boldsymbol{0}}$ and $\rho|_{\boldsymbol{1}} = \rho_{\boldsymbol{1}}$. Since $\rho_{\boldsymbol{0}}$ and $\rho_{\boldsymbol{1}}$ are unique, so is ρ. By Theorem 5, $(\mathcal{H}_f^\tau \cup S)^* = \mho^\tau(u)$. □

Theorem 16. *Algorithm 2 is correct; that is, $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{B})$.*

Proof. The following statements are equivalent by Lem. 15 and Cor. 6: **(1)** $t \in \mathcal{L}\mathrm{ng}(\mathcal{A})$ **(2)** there is a loop $(q_\mathrm{i}, q_\mathrm{f}) \in I \times F$ of $\mathcal{A}$ on t **(3)** the run ρ of $\mathcal{B}$ on t is such that $\rho(\varepsilon) = (t(\varepsilon), \star, \mho^\star(t))$, with $(q_\mathrm{i}, q_\mathrm{f}) \in \mho^\star(t)$ **(4)** $\rho(\varepsilon)$ is a final state for $\mathcal{B}$ **(5)** $t \in \mathcal{L}\mathrm{ng}(\mathcal{B})$. □

Two short but important remarks are in order. First: it might seem strange that our states are in $\Sigma \times \mathbb{T} \times 2^{Q^2}$, and not more simply in $\mathbb{T} \times 2^{Q^2}$, as suggested in [10]. In [9] a similar construction – albeit deterministic, see the second remark – is proposed, which does not include Σ either. However, it is not clear how loops could be considered independently from the root symbol of the subtree that bears them. Consider for instance $a, b \in \Sigma_0$ with only the transitions $\langle \{ a, b \}, p, \tau \to \circlearrowleft, q \rangle$ and $\langle b, q, \tau \to \uparrow, s' \rangle \in \Delta$. Then the loops on a and b are exactly the same – $\{(p, q)\}^*$ – and yet, from their father's point of view, they behave very differently. If $\mathcal{A}$ can go down from a state s to p, it can form a loop (s, s') if the child is b, but not if it is a. In contrast to the loop-based construction, the overloop-based algorithm (Algo. 3) suppresses this problem completely.

Second: the observation made in Lemma 15 that the run of $\mathcal{B}$ is unique, given a subtree and a type, makes it easy to adapt the algorithm to yield a deterministic BUTA. Indeed, every tree in $\mathcal{T}(\Sigma)$ is non-deterministically evaluated by $\mathcal{B}$ into exactly three possible states (one per type); the correct one is chosen according to the context during the run. Recall that rules $f(P_{\boldsymbol{0}}, P_{\boldsymbol{1}}) \to P$ are built such that the "type" component of P_θ is θ, and final states bear the root type $\star$. Hence, it

suffices to group those three possible states into one element of $\Sigma \times (2^{Q^2})^{|\mathbb{T}|}$ to achieve determinism[3], which brings us back to the states suggested in [9].

Lemma 17 (*Overloop-Based Algorithm*)**.** *Let $\mathcal{A}$ be a TWA, $\mathcal{B}$ the BUTA constructed by Algorithm 3, $t \in \mathcal{T}(\Sigma)$ and a position $\alpha \in \mathcal{P}os(t)$. Then for every type $\tau \in \mathbb{T}$ there is a unique run ρ of $\mathcal{B}$ on $t|_\alpha$, which is such that $\rho(\varepsilon) = (\tau, \mathfrak{G}^\tau(t|_\alpha))$.*

Proof. See proof of Lemma 15. The only change is that this time, we build the loops, then deduce the overloops from them (Lem. 11[p7], Thm. 12). □

Theorem 18. *Algorithm 3 is correct; that is, $\mathcal{L}ng(\mathcal{A}) = \mathcal{L}ng(\mathcal{B})$.*

Proof. By construction $(i, \checkmark) \in I \times \{\checkmark\}$ is an overloop iff there exists $f \in F$ such that (i, f) is a loop. Same proof as Theorem 16. □

Note that this construction can be adapted to yield deterministic BUTA in exactly the same way as for Algo. 2.

4.2. Overloops and the Deterministic Case: an Upper-Bound for the Size

Definition 19 (Deterministic TWA)**.** A TWA $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ is *deterministic* (ie. a *DTWA*) if[4] $|\langle \sigma, p, \tau \to \mathbb{M}, Q \rangle \cap \Delta| \leqslant 1$ for all $\sigma \in \Sigma$, $p \in Q$, $\tau \in \mathbb{T}$.

Example: The running example TWA $\mathcal{X}$ happens to be a deterministic TWA.

Definition 20 (Functional Relation)**.** A relation $R \subseteq Q^2$ is *functional* (or *right-unique*, or *a partial function*) if, for all $p, q, q' \in Q$, pRq and $pRq' \implies q = q'$.

Remark 21. There are $2^{|Q|^2}$ binary relations on Q, of which $|Q+1|^{|Q|}$ are partial functions, of which $|Q|^{|Q|}$ are total functions.

Remark 22. If a relation R is functional, then so is R^k, for any $k \in \mathbb{N}$.

By construction, a BUTA built by Algo. 2 (loop-based) has at most $|\Sigma| \cdot |\mathbb{T}| \cdot 2^{|Q|^2}$ states, while one built by Algo. 3 (overloop-based) has at most $|\mathbb{T}| \cdot 2^{|Q|^2}$. We will see in this section that, in the deterministic case, this upper bound is in fact much lower for the overloop-based algorithm than for the traditional loop-based one. More specifically, we will show that the following holds:

Theorem 23 (*Deterministic Upper-Bound*)**.** *Let $\mathcal{A}$ be a deterministic TWA and $\mathcal{B}$ its equivalent BUTA built by an application of Algorithm 3. Then $\mathcal{B}$ has at most $|\mathbb{T}| \cdot 2^{|Q| \log_2(|Q|+1)}$ states.*

---

[3] Of course, if one does that, there are a number of optimisations which can be performed. For instance, since the star-component is only ever useful at the root, it suffices to replace it with a boolean indicating whether it contains a loop in $I \times F$, i.e. whether it is a final state. Then we get states in $\Sigma \times (2^{Q^2})^{|\mathbb{S}|} \times \{0, 1\}$.

[4] In this paper we do not need the usual, stronger definition, where I is a singleton.

The idea is that every state which we build corresponds exactly to the set L of all loops (resp. overloops) of the automaton $\mathcal{A}$ on a certain subtree u. Since $L \subseteq Q^2$, we can see it as a binary relation on the states. The intuition here is that, if $\mathcal{A}$ is deterministic, and enters the root of u in one given state p, then there "should be" only one possible outcome. More formally:

Lemma 24. *If $\mathcal{A}$ is a deterministic TWA, then $\twoheadrightarrow_{\mathcal{A}}$ is functional.*

Proof. In a given configuration (α, p), over a tree t, $|\langle t(\alpha), p, \natural\alpha \to \mathbb{M}, Q\rangle \cap \Delta|$ $\leqslant 1$. Therefore, (α, p) has at most one successor. □

However, in the case of loops, this does not suffice to make L functional because, determinism notwithstanding, a single (non-trivial) loop may reach the root several times, and in different states, before exiting the subtree. Thus there is nothing to prevent us from having both pLq and pLq', for $q \neq q'$; we show next that in that case, one of these loops is simply an extension of the other.

Lemma 25 (*Hidden Loops*)**.** *Let (p, q) and (p, q') be loops of the TWA $\mathcal{A}$ on a given subtree $t|_\alpha$, such that $q \neq q'$. Then if $\mathcal{A}$ is deterministic, either (q, q') or (q', q) must be a loop of $\mathcal{A}$ on $t|_\alpha$.*

Proof. By Definition 1, there exist two runs $c_0, \ldots, c_n$ and $d_0, \ldots, d_m$ such that $c_0 = d_0 = (\alpha, p)$, $c_n = (\alpha, q)$ and $d_m = (\alpha, q')$. If $n = m$ then $c_0 \twoheadrightarrow^n c_n$ and $c_0 \twoheadrightarrow^n d_n$ and by Lemma 24 and Remark 22, it follows that $c_n = d_m$. But this contradicts $q \neq q'$, so we must have $n \neq m$. Say that $n < m$. Then $c_n = d_n$, and $(\alpha, q) = d_n, \ldots, d_m = (\alpha, q')$ forms a run. Therefore (q, q') is a loop. Similarly, if $n > m$, then by the same arguments (q', q) is a loop. □

Contrariwise, two overloops cannot be combined to form another overloop on the same subtree, which satisfies the above intuition of a "single outcome":

Lemma 26. *Let $p, q, q' \in Q$, such that (p, q) and (p, q') are overloops of the TWA $\mathcal{A}$ on a given subtree $t|_\alpha$. Then if $\mathcal{A}$ is deterministic, $q = q'$.*

Proof. By Def. 9, there exist $s, s' \in Q$ such that $(\alpha, p), \ldots, (\alpha, s), (\overline{\mathfrak{p}}(\alpha), q)$ and $(\alpha, p), \ldots, (\alpha, s'), (\overline{\mathfrak{p}}(\alpha), q')$ are runs; thus (p, s) and (p, s') are loops. If $s \neq s'$, then by Lem. 25, say, (s, s'), is a loop. So there exist $s_1, \ldots, s_n \in Q, \beta_1 \trianglelefteq \alpha$ $, \ldots, \beta_n \trianglelefteq \alpha$ such that $(\alpha, s), (\beta_1, s_1), \ldots, (\beta_n, s_n), (\alpha, s')$ is a run. Thus we have in particular $(\alpha, s) \twoheadrightarrow (\overline{\mathfrak{p}}(\alpha), q)$ and $(\alpha, s) \twoheadrightarrow (\beta_1, s_1)$. It follows that $\overline{\mathfrak{p}}(\alpha) = \beta_1 \trianglelefteq$ α, which is contradictory. Hence $s = s'$. We have both $(\alpha, s) \twoheadrightarrow (\overline{\mathfrak{p}}(\alpha), q)$ and $(\alpha, s) \twoheadrightarrow (\overline{\mathfrak{p}}(\alpha), q')$. Since $\twoheadrightarrow$ is functional (Lem. 24), we have finally $q = q'$. □

With this, we can conclude the proof of Theorem 23.

Proof of Theorem 23. By construction, for every state $P = (\tau, L)$ generated for $\mathcal{B}$ by Algorithm 3, there exists at least a subtree t such that L is the set of overloops of $\mathcal{A}$ on t. Thus, by Lemma 26, L is functional. Therefore, by Remark 21, there are at most $|\mathbb{T}| \cdot |Q + 1|^{|Q|}$ states (or, equivalently, $|\mathbb{T}| \cdot 2^{|Q| \log_2(|Q|+1)}$). □

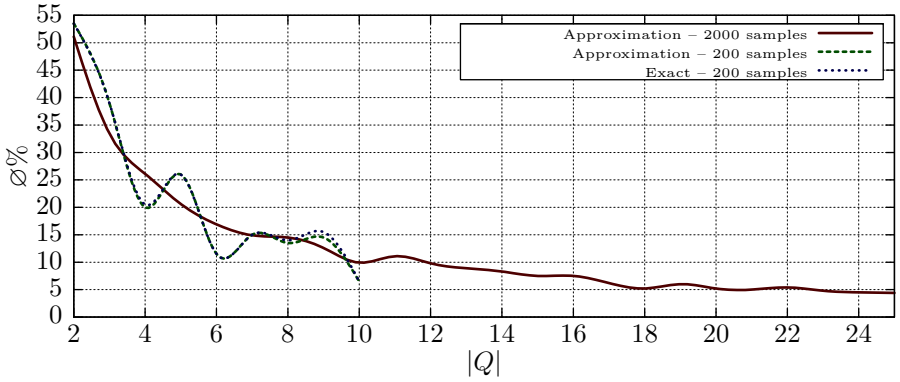

Figure 1: Uniform random TWA: Emptiness results

Note that the same bound (with a $|\Sigma|$ factor) might be achievable using loops, if special provisions are made to determine which of two loops (p, q) and (p, q') subsumes the other, and to remove the superfluous loops from the states as they are built. However, such provisions would be invalid if $\mathcal{A}$ is not deterministic, unlike the overloops method, which is applicable in all generality.

5. A Polynomial Over-Approximation for the Emptiness Problem

Data: An escaped TWA $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ (see Def. 13)
Result: Empty (only if $\mathcal{L}\mathrm{ng}(\mathcal{A}) = \varnothing$) or Unknown

initialise $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_\star$ to $\varnothing$; **foreach** $a \in \Sigma_0, \tau \in \mathbb{T}$ **do** $\mathcal{L}_\tau \leftarrow \mathcal{L}_\tau \cup \mathcal{U}_a^\tau[\mathcal{H}_a^{\tau *}]$
repeat
  foreach $f \in \Sigma_2, \tau \in \mathbb{T}$ **do** $\mathcal{L}_\tau \leftarrow \mathcal{L}_\tau \cup \mathcal{U}_f^\tau\left[(\mathcal{H}_f^\tau \cup S)^*\right]$
  where $S = \left\{ (p, q_\theta) \;\middle|\; \exists \theta \in \mathbb{S}, p_\theta \in Q : \middle| \begin{array}{c} \langle f, p, \tau \to \chi(\theta), p_\theta \rangle \in \Delta \\ \text{and } (p_\theta, q_\theta) \in \mathcal{L}_\theta \end{array} \right\}$
until $\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_\star$ *remain unchanged*
return Empty *if* $\mathcal{L}_\star \cap (I \times \{\checkmark\}) = \varnothing$, *else* Unknown

Algorithm 4: Approximation for emptiness, with overloops

Testing emptiness of a TWA $\mathcal{A}$ is an ExpTime-complete problem [9]. This is rather unfortunate, as there are practical questions – such as satisfiability of some XPath fragments – which reduce to the emptiness of the language of a TWA. We present in this section a crude but fairly accurate and very expeditious overloops-based algorithm capable of detecting emptiness in a number of cases. Algorithm $4_{[\mathrm{p13}]}$ is a variant of Algorithm 3 with the following properties:

Lemma 27 (*Overloops Over-Approximation*)**.** *Let $\mathcal{A}$ be a TWA; when the execution of Algorithm 4 ends, then for any $\tau \in \mathbb{T}$, $\mathcal{L}_\tau \supseteq \bigcup_{t \in \mathcal{T}(\Sigma)} \mathfrak{G}^\tau(t)$.*

Proof. This result is fairly clear when comparing Algorithms 3 and 4. Let us consider a tree t and a subtree $u = t|_\alpha$, with $\tau = \natural\alpha$. We show that $\mathfrak{O}^\tau(u) \subseteq \mathcal{L}_\tau$. *Base case:* $u = a \in \Sigma_0$. Then by the first line of Algo. 4, we have $\mathfrak{O}^\tau(a) = \mathcal{U}_a^\tau[\mathcal{H}_a^{\tau *}] \subseteq \mathcal{L}_\tau$. *Inductive case:* If $u = f(u_0, u_1), f \in \Sigma_2$, then by induction hypothesis we have $\mathfrak{O}^0(u_0) \subseteq \mathcal{L}_0$ and $\mathfrak{O}^1(u_1) \subseteq \mathcal{L}_1$. The expression computed in the main loop is almost the same as that of Thm. 12 for $\mathfrak{O}^\tau(u)$, the only difference being that $\mathcal{L}_\theta$ is used instead of $\mathfrak{O}^\theta(u_\theta)$. Since we have $\mathfrak{O}^\theta(u_\theta) \subseteq \mathcal{L}_\theta$ for all $\theta \in \mathbb{S}$, the expression in Algo. 4 computes *at least* all overloops of $\mathfrak{O}^\tau(u)$ — and adds them to $\mathcal{L}_\tau$. Thus $\mathfrak{O}^\tau(u) \subseteq \mathcal{L}_\tau$. □

Theorem 28. *Algorithm 4 is correct; that is, it yields* Empty *only if* $\mathcal{L}ng(\mathcal{A}) = \varnothing$.

Proof. Suppose that Algo. 4 yields Empty. By definition, this is the case if and only if $\mathcal{L}_\star \cap (I \times \{\checkmark\}) = \varnothing$. By Lemma 27, we have $\bigcup_{t \in \mathcal{T}(\Sigma)} \mathfrak{O}^\tau(t) \subseteq \mathcal{L}_\tau$ for all types τ, and it follows that in particular $(\bigcup_{t \in \mathcal{T}(\Sigma)} \mathfrak{O}^\star(t)) \cap (I \times \{\checkmark\}) = \varnothing$. This can be equivalently rephrased as $\forall t \in \mathcal{T}(\Sigma)$, $\mathfrak{O}^\star(t) \cap (I \times \{\checkmark\}) = \varnothing$. By Corollary 14, this translates into: for all $t \in \mathcal{T}(\Sigma), t \notin \mathcal{L}\mathrm{ng}(\mathcal{A})$, that is to say, $\mathcal{L}\mathrm{ng}(\mathcal{A}) = \varnothing$. □

Corollary 29 (*Complexity of the Approximation*)**.** *The execution of Algorithm 4 is done in time polynomial in the size of* $\mathcal{A}$ *— more precisely:* $O(|\Sigma| \cdot |\mathbb{T}|^2 \cdot |Q|^5)$.

Proof. For all types τ, all operations in Algo. 4 which alter $\mathcal{L}_\tau$ add elements to it. The first loop executes a fixed number of times ($|\Sigma_0| \times |\mathbb{T}|$). The main loop contains only an inner loop which executes a fixed number of times as well ($|\Sigma_2| \times |\mathbb{T}|$), and the main loop itself executes until no element is added to $\mathcal{L}_0$, $\mathcal{L}_1$ or $\mathcal{L}_\star$ during the iteration. Since an iteration can only add elements, and each iteration adds at least one, there can be at most

$$\sum_{\tau \in \mathbb{T}} |\mathcal{L}_\tau| = \sum_{\tau \in \mathbb{T}} |Q|^2 = |\mathbb{T}| \times |Q|^2$$

iterations of the main loop. Each iteration of both the first loop and the main inner loop computes a set of overloops, based on two sets of previously-computed (potential) overloops. This operation executes in a time which is in time $O(|Q|^2 \cdot |\Delta|)$ for the initial computation and $O(|Q|^3)$ for the computation of the transitive closure. It is executed in total $|\Sigma_0| \cdot |\mathbb{T}| + |\mathbb{T}| \cdot |Q|^2 \cdot (|\Sigma_2| \cdot |\mathbb{T}|)$ times. Overall, the number of executions is in $O(|\Sigma| \cdot |\mathbb{T}|^2 \cdot |Q|^2)$. Globally, the execution time of Algo. 4 is in $O(|\Sigma| \cdot |\mathbb{T}|^2 \cdot |Q|^5)$. This is of course a *very* loose bound. □

Note that Algorithm 4 can easily be made just as coarse or as fine as the need dictates. At the coarse end of that gamut we have a variant of Algorithm 4 which forgoes type information, thus hoarding up all overloops in a single set $\mathcal{L}$ instead of three, and at the fine end we find something equivalent to Algorithm 3.

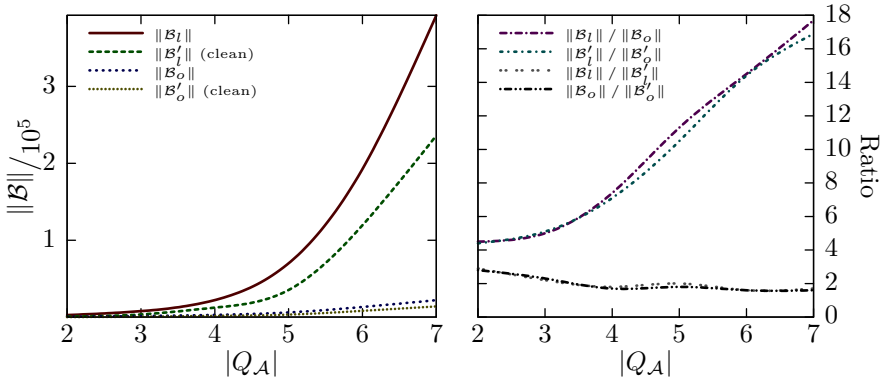

Figure 2: Uniform random TWA: Size results

6. Experimental Results

6.1. Evaluating the Approximation's Effectiveness

Tests have been conducted against two different sets of randomly generated TWA. The first set comprised roughly twenty thousand random automata of various sizes ($2 \leqslant |Q| \leqslant 20$), with a small number of rules ($|\Delta| \approx 3 \times |Q|$) and the same alphabet as for our running example $\mathcal{X}$. The random generation scheme which produced them was *ad-hoc* and did not have any pertinent statistical grounds. The approximation yielded astonishingly good results on this set: about 75% of the automata had empty languages, yet the approximation failed to detect emptiness in only two cases. To confirm those encouraging results, we generated a second set of (complete and deterministic[5]) TWA, this time according to a uniform probability distribution [11]. The `REGAL` library [16] was used as back-end to generate the underlying finite-state automata. More specifically, 2 000 TWA were uniformly generated for each $|Q|$ within the range $2 \leqslant |Q| \leqslant 25$. Figure $1_{[\text{p13}]}$ summarises the performance of the approximation on this set. The first curve presents the percentage of TWA whose language is detected to be empty by the approximation among the whole 2 000 TWA, for each $|Q|$. The second curve presents the same results, but only for the first 200 TWA[6] for each $|Q| \leqslant 10$; the third curve presents the *exact* results for the same data as the second. It is visible that the approximation performs very well again, as the second and third curves are almost indistinguishable. Out of the 1 724 TWA for which both the approximation and an exact algorithm were run, of which 398 had empty languages, only four failures of the approximation were observed. Furthermore, the first curve shows that the approximation continues to catch

[5] Note that $|Q|$ is therefore proportional to the size of the generated TWA.

[6] With the exception of the last data point (namely, $|Q| = 10$), for which only the first 124 TWA were tested; this is due to both time constraints and memory limitations of the computer used for the tests.

cases of emptiness even for sizes completely intractable with exact algorithms. Those results, though statistically sound, are probably much better than what can be expected in practical applications; it is likely that random instances are in some sense trivial wrt. emptiness. In the absence of substantial testbeds from real-world applications of TWA, a study similar to [12] could be conducted to flesh out the properties which would make an instance "difficult" wrt. emptiness.

6.2. Overloops Yield Smaller BUTA

Comparing the output of Algos. 2 & 3, we noted that the latter generates smaller automata. By way of example, if $\mathcal{B}_\mathrm{l}$ is the equivalent BUTA obtained from $\mathcal{X}$ by Algo. 2, and $\mathcal{B}_\mathrm{o}$ by Algo. 3, then we have $\|\mathcal{B}_\mathrm{l}\| = 1986$ and $\|\mathcal{B}_\mathrm{o}\| = 95$, where the size of a BUTA $\mathcal{B} = \langle \Sigma, Q, F, \Delta \rangle$ is defined – in the usual way [14] – as:

$$\|\mathcal{B}\| \triangleq |Q| + \sum_{f(p_1,\ldots,p_n)\to q \in \Delta} \big(n+2\big) \ .$$

Note that the resulting automata are quite large, even for such a trivial TWA as $\mathcal{X}$! For comparison, consider the manually constructed (deterministic) minimal BUTA $\mathcal{B}_\mathrm{m}$, and the[7] smallest possible non-deterministic BUTA $\mathcal{B}_\mathrm{s}$ equivalent to the TWA $\mathcal{X}$: we have $\|\mathcal{B}_\mathrm{m}\| = 56$ and $\|\mathcal{B}_\mathrm{s}\| = 34$. In other words, the loops and overloops-based constructions are about three and sixty times larger than the optimal, respectively. More important than the size of the final BUTA is the computation time; it just happens in practice to be roughly proportional to the size of the result, as far as our two transformations are concerned. Using a deterministic variant of either transformation and minimising the result would yield $\mathcal{B}_\mathrm{m}$, but at the cost of a considerable increase of the worst-case complexity and average computation time. Another important point is that the huge size discrepancy between $\mathcal{B}_\mathrm{l}$ and $\mathcal{B}_\mathrm{o}$ cannot be reduced "in post-processing" using the standard BUTA reduction (elimination of unreachable states [14]): it would have no effect whatsoever, because Algorithms 2 and 3 yield reduced BUTA by construction. A more powerful operation such as the `cleanup` method described in [12], which removes states which are not co-accessible as well as unreachable states, can bring down the sizes of $\mathcal{B}_\mathrm{l}$ and $\mathcal{B}_\mathrm{o}$, but does in no way bridge the gap between them. Case in point, the automata after cleanup $\mathcal{B}'_\mathrm{l}$ and $\mathcal{B}'_\mathrm{o}$ are of sizes[8] $\|\mathcal{B}'_\mathrm{l}\| = 1617$ and $\|\mathcal{B}'_\mathrm{o}\| = 78$, which yield the following ratios:

$$\frac{\|\mathcal{B}_\mathrm{l}\|}{\|\mathcal{B}_\mathrm{o}\|} \approx 20.9 \quad \text{and} \quad \frac{\|\mathcal{B}'_\mathrm{l}\|}{\|\mathcal{B}'_\mathrm{o}\|} \approx 20.7 \quad \text{and} \quad \frac{\|\mathcal{B}_\mathrm{l}\|}{\|\mathcal{B}'_\mathrm{l}\|} \approx \frac{\|\mathcal{B}_\mathrm{o}\|}{\|\mathcal{B}'_\mathrm{o}\|} \approx 1.2 \ .$$

These figures suggest that the (substantial) size gains originating from the switch from loops to overloops-based algorithms are completely unrelated to (and do not

[7] It happens to be unique (up to homomorphism) in this particular case.

[8] In this trivial example, the sizes of the TWA $\mathcal{X}$, of the "optimal" equivalent BUTA $\mathcal{B}_\mathrm{m}$ and $\mathcal{B}_\mathrm{s}$, and of the post-cleanup overloops-based BUTA $\mathcal{B}'_\mathrm{o}$ happen to be quite close. This observation should of course not be generalised.

interfere with) the (modest) size gains from post-processing. The observations drawn from this single example have been substantiated by more thorough experiments conducted on the same uniformly generated random TWA as in Fig. 1, the results of which are summarised in Fig. $2_{[p15]}$. The legend uses the same notations as above. Two hundred TWA have been used to construct each data point.[9]

6.3. Demonstration Software

Readers interested in experimenting with this paper's algorithms will find online[10] a proof of concept (binaries and `OCaml` source code), as well as comprehensive instructions for use.

7. Conclusion

In this paper we have introduced tree overloops, and applied both loops and overloops to common operations on TWA: deciding membership, transforming a TWA into a BUTA, and inexpensively testing emptiness. We have shown that the use of overloops simplifies the transformation into BUTA, and substantially lowers the upper bound in the deterministic case. We intend to pursue this further by using overloops to characterise useful classes of TWA and perform significant simplifications on the automata, hopefully leading to applications to XPath. Furthermore, while our theoretical results and experiments show that the overloops-based transformation yields much smaller BUTA than the loops-based one, both asymptotically and in average – and yields them proportionally faster, – it is clear that further advances remain possible in that respect. On-the-fly variants enabling to test emptiness (for instance) while forgoing the computation of the whole BUTA would also be of interest.

Acknowledgements. The authors would like to thank the members of the INRIA ARC ACCESS for interesting discussions on this topic. Our thanks go as well to the anonymous reviewers – for both the conference version and the journal version – who provided the tighter complexity bound for Cor. 7, and whose careful proofreading improved the readability of this paper.

References

[1] A. Aho, J. Ullman, Translations on a context free grammar, Information and Control 19 (1969) 439–475.

[2] B. ten Cate, L. Segoufin, Transitive closure logic, nested tree walking automata, and XPath, J. ACM 57 (2010) 251–260.

[9] The same remark as for Fig. 1 applies: Fig. $2_{[p15]}$ uses only 156 TWA for its last data point ($|Q| = 7$).

[10] On http://lifc.univ-fcomte.fr/~vhugot/TWA.

[3] L. Segoufin, V. Vianu, Validating Streaming XML Documents, in: PODS, ACM, 2002, pp. 53–64.

[4] M. Bojańczyk, T. Colcombet, Tree-walking automata do not recognize all regular languages, STOC '05, ACM, 2005, pp. 234–243.

[5] M. Bojańczyk, 1-bounded TWA cannot be determinized, FSTTCS'03, LNCS 2914 (2003) 62–73.

[6] M. Bojańczyk, T. Colcombet, Tree-walking automata cannot be determinized, Theoretical Computer Science 350 (2006) 164–173.

[7] D. Knuth, Semantics of context-free languages, Theory of Computing Systems 2 (1968) 127–145.

[8] J. Shepherdson, The reduction of two-way automata to one-way automata, IBM Journal of Research and Development 3 (1959) 198–200.

[9] M. Bojańczyk, Tree-Walking Automata, LATA'08 (tutorial), LNCS 5196 (2008).

[10] M. Samuelides, Automates d'arbres à jetons, Ph.D. thesis, Université Paris-Diderot - Paris VII, 2007.

[11] P.-C. Héam, C. Nicaud, S. Schmitz, Random Generation of Deterministic Tree (Walking) Automata, LNCS 5642 (2009) 115–124.

[12] P. Heam, V. Hugot, O. Kouchnarenko, Random Generation of Positive TAGEDs wrt. the Emptiness Problem, Technical Report RR-7441, INRIA, 2010.

[13] P.-C. Héam, V. Hugot, O. Kouchnarenko, Loops and overloops for tree walking automata, in: CIAA'11, LNCS 6807, pp. 166–177.

[14] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, 2007.

[15] S. Cosmadakis, H. Gaifman, P. Kanellakis, M. Vardi, Decidable optimization problems for database logic programs, STOC '88, ACM, 1988, pp. 477–490.

[16] F. Bassino, J. David, C. Nicaud, `REGAL` : A library to randomly and exhaustively generate automata, in: CIAA, LNCS 4783, pp. 303–305.